POLYTECHNIC UNIVERSITY OF CATALONIA (UPC)
UNIVERSITY OF AUCKLAND (UOA)

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Automatic Ventricle Segmentation Using CNNs in Cardiac MRI

Author:
Carles Garcia Cabrera

Supervisor:
Prof. Alistair A. Young

Examiner:
Prof. Verónica Vilaplana Besler

Co-supervisor:
Dr. Pau Medrano-Gracia

in the
Faculty of Medical and Health Sciences
Auckland MRI Research Group

January 24, 2019
Auckland, New Zealand

POLYTECHNIC UNIVERSITY OF CATALONIA (UPC)
UNIVERSITY OF AUCKLAND (UOA)

Abstract

Faculty of Medical and Health Sciences
Auckland MRI Research Group

by Carles Garcia Cabrera

Cardiac magnetic resonance imaging has been proven to be a great aid tool in clinical diagnosis. Computational models arising from these images have been developed for many years by engineers, radiologists and clinicians. A first task in this process is to segment the different regions of the heart, where machine learning and, more recently, deep learning, have shown good performance. My project aims to improve the current network performance when segmenting the left-ventricular, myocardial and right-ventricular regions through (1) data augmentation, (2) data-set combination and (3) loss-function optimization, with a limited amount of computational resources. Results show improvements for all three methodologies. In addition, investing computational resources on muscular regions provides better performance in cavity regions.

Acknowledgements

I would like to thank my project supervisor Prof. Alistair A. Young and my co-supervisor Dr. Pau Medrano-Gracia from the MRI Research Group at the University of Auckland for their trust and help. Thanks also to Dr. Conrad Werkhoven for helping me with their deep neural network model, and also the previous interns who contributed to its development: Callum Herries, Tim Hermans, Thibault Escobar and Tim Finucane.

I would also like to thank Prof. Óscar Cámara for his help during the project, and for sharing a lot of interesting information with me. It has been really motivating to hear him talk about his projects and his perspective on the current state-of-the-art of bio-medical engineering.

Thanks also to Edward Ferdian for his advice and support, it has been a really rewarding experience to talk about the new machine learning techniques and papers with him. I am also thankful to my other colleagues of the MRI Research Group for their warm welcome and friendship.

I would like to thank also Prof. Ferran Marqués for the trust and courage given to me. Thanks also to Prof. Verónica Vilaplana for agreeing to be my administrative tutor and examiner.

Last but not least, thanks to my family and friends for encouraging me to pursue my dreams and helping me while doing so, especially my grandparents for supporting my expenses during this experience abroad and for being such an excellent example for me.

Contents

List of Figures

List of Tables

Abbreviations

CNN	**C**onvolutional **N**eural **N**etwork
ED	**E**nd **D**iastole
ES	**E**nd **S**istole
CHD	**C**ardiac **H**eart **D**isease
NN	**N**eural **N**etwork
UKB	**U**nited **K**ingdom **B**iobank
RV	**R**ight **V**entricle
LV	**L**eft **V**entricle
MYO	**M**yocardium
MRI	**M**agnetic **R**esonance **I**maging
MR	**M**agnetic **R**esonance
CMR	**C**ardiac **M**agnetic **R**esonance
TP	**T**rue **P**ositive
FP	**F**alse **P**ositive
FN	**F**alse **N**egative
TN	**T**rue **N**egative
SPP	**S**patial **P**yramid **P**ooling
SGD	**S**tochastic **G**radient **D**escent
GT	**G**round **T**ruth
SNR	**S**ignal **N**oise **R**atio

Chapter 1

Introduction

The following dissertation ("Treball de Final de Grau", `TFG`) summarizes my internship in the Auckland MRI research group. We looked to apply Artificial Intelligence algorithms to improve the performance of the current methods for segmenting cardiac MR images. This segmentation, together with statistical models, can be used to diagnose cardiac diseases such as **Ventricular Hypertrophy**.

1.1 Objective

The scope of the project is to segment cardiac short axis MRI images from a dataset called **SAVE** into Right Ventricle Cavity, Left Ventricle Cavity and Myocardium. The projects aims to predict masks over images from different scanners, dealing with different image shapes. To achieve that, Convolutional Neural Networks will be used, in this particular case, the U-net network [1] will be the starting point. Also, its hyperparameters, limitations and strengths will be analyzed.

As a secondary objective, we would like to reduce the training and prediction time. Also, we would like the be able to re-train over an already trained model and prepare the network to serve a secondary network following the concept of *transferred learning*.

After the first results we are going to consider applying other techniques or changes in the network (or in the model) to improve the results and performance.

Manual segmentation of Cardiac Magnetic Resonance images is a very tedious process and differences arise between different analysts (inter-observer variability). The aim of an automatic segmentation is to perform segmentation over a set of images, quickly, and with consistency, allowing this information to be used later on for diagnosis, for example.

The segmentation of Cardiac Magnetic Resonance images finds its applications in various cases. For example, the identification of contours in a CMR image allows for automatic clinical measurements. Also, the segmentation of the images of several slices enables the creation of 3D models of the heart. If, for each slice, we obtain several images during one or more heartbeats, it is also possible to perform 4D (3D + time) modeling of the heart, and then evaluate its motion and even attempt to pre-diagnose some heart conditions, which are among the top killers worldwide, contributing to 30% of total global deaths [2].

1.2 Environment

We used `Windows` [3] and `Anaconda` [4] as the virtual environment manager, and `Visual Studio Code` [5] as a development tool. The whole project was written in `Python` [6] but some of the pre-processing was done in `Matlab` [7].

Regarding other packages that we used, `TensorFlow` [8] and `OpenCV2` [9] are the most important due to their powerful handling of Deep Learning and Image Processing respectively [10, 11]. We also considered testing concepts and performances using `Keras` [12] due to its suitability for Deep Neuronal Networks [13].

All experiments except the pre-processing were run in a server with a `NVIDIA Titan X` [14].

Tests using `Keras` were performed (when needed) in another server with a `NVIDIA Titan X` using a `Jupyter notebook` [15].

1.3 Constrains and Limitations

The project was built over an existing network and model, providing some constrains while changing parameters and architectures. To avoid compatibility problems, most parts were developed to be patch-like, ensuring that the original features of the network would work even if they could be seen as out-of-date for our purpose.

The original architecture faces the problem that is not compatible with all sizes. At the beginning of the project the models managed to up-sample when images were smaller than 256x256 pixels but was unable to process bigger images. Two different methods were proposed to solve this: using down-sampling and using Pyramidal Pooling in the architecture [16].

Even though our amount of data is quite big, we faced the problem that our prediction data-set did not have any ground-truth nor a golden standard, preventing the use of immediate metrics. For the purposes of this project, expert analysts helped determine whether the results were qualitatively acceptable or not.

Chapter 2

Background

In this section we present previous work, with the aim of building on and improving the current prediction system.

As this project is focused on machine learning applied to CMR segmentation, we firstly review some of the main key clinical concepts of MRI. Secondly, we review the state-of-the-art of machine learning.

2.1 Previous Projects

At the AMRG research group, several attempts to segment CMR images have been explored, mostly in the LV, and one for the RV. However due to the limitation of the network's input size, previous attempts resulted in over-fitting, and thus sub-optimal results. While LV results might have been acceptable for myocardial segmentation, these were deemed not acceptable for the RV.

Based on suggested future work from one of these earlier projects, we would like to (1) explore the further adjustment of weights, in particular that of the mean *DICE* score, to improve learning, and (2) use data augmentation to decrease the over-fitting.

2.2 Basic Cardiac Anatomy

Since ground truth data was not available at the beginning of this project, we had to determine, with the analysts' help, whether the predictions were acceptable. This requires some cardiac anatomical and clinical background, and extensive post-processing work.

We focus on the most clinically relevant chambers of the heart 2.1 and 2.2), namely the Right and Left Ventricle [17]. It is important to notice that the reader would see the ventricles as if they were facing the patient (the famous "left-right swap" rule in radiology).

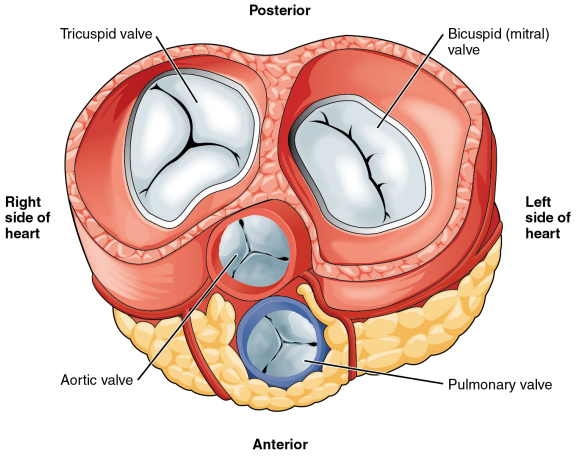

FIGURE 2.1: Short Axis Cut

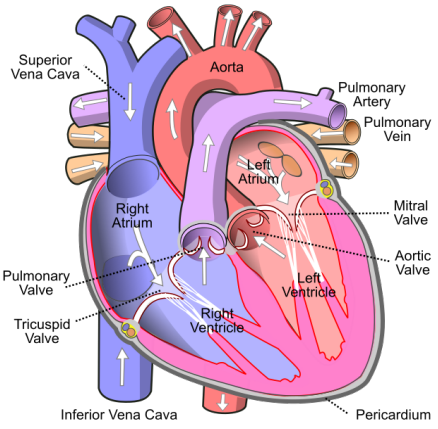

FIGURE 2.2: Long Axis Cut

The myocardium is an involuntary signal-paced striated muscle that constitutes the main tissue of the walls of the heart. The myocardium is an important focus of our work, as it represents the boundary of the chambers.

Between the epicardium (outer layer of the heart wall) and the endocardium (inner layer of the heart wall) there is a thick middle layer called myocardium, its blood is supplied via coronary circulation. Myocardial cells are called cardiomyocytes (heart muscle cells) and are joined together by intercalated discs. The matrix encasing them with collagen fibres and other substances is called the extracellular matrix [18].

2.3 Magnetic Resonance Imaging

In this section we present a very brief overview of MRI image generation, paying special attention to the parameters related to resolution and SNR.

The images are obtained by applying the *Fourier Transform* to K-space. The K-space is where samples obtained while scanning are stored, and it has its own spatial resolution, as seen in 2.3 and following the Frequency Resolution and Spatial Resolution [19].

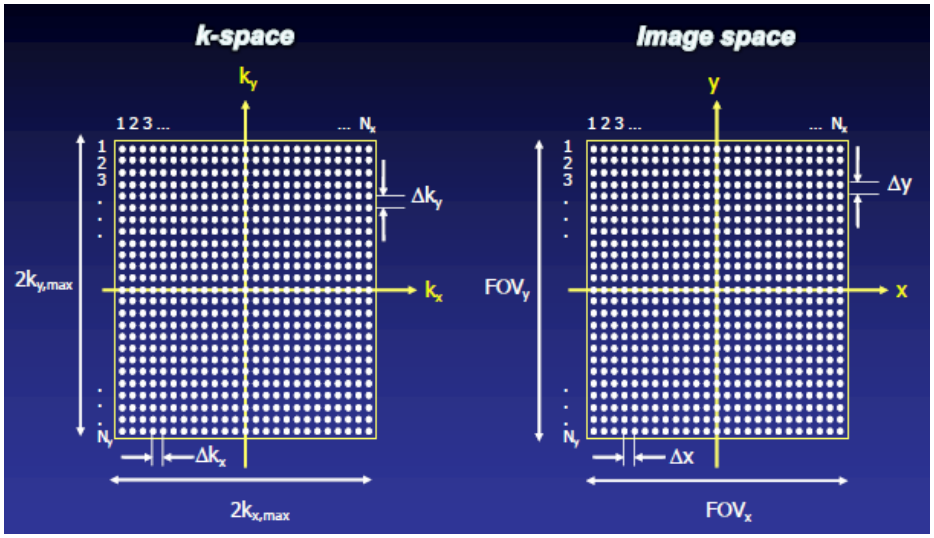

FIGURE 2.3: Association between Image Space and k-Space

The scanning time has a strong impact in the quality of the resulting image as we can see in Scan Time SNR Tradeoff [19], where NEX is the number of signal averages, B_0 a constant when working with MR and TR is the repetition time used to obtain the samples. Therefore, to reduce scanning time, sometime images are up-sampled.

$$\Delta k_x = \frac{1}{FOV_x} \quad \text{(Frequency Resolution)} \qquad \Delta_x = \frac{1}{2K_{x,max}} \quad \text{(Spatial Resolution)}$$

$$ScanTime = N_y \times TR \times NEX \quad \text{(Scan Time)}$$

$$SNR = \frac{B_0 \times \sqrt{NEX} \times SliceThickness \times FOV^2}{\sqrt{N_x} \times \sqrt{N_y} \times \sqrt{ReceiverBandwidth}} \quad \text{(SNR Tradeoff)}$$

2.3.1 Scanners

- **SIEMENS Aera**: 1.5 T scanner. Images in the SAVE set are 156x192 and 240x196.
- **SIEMENS Avanto**: 1.5 T scanner. Images in the SAVE set are 216x256.
- **GE Signa HDxt**: 3 T scanner. Images in the SAVE set are 256x256.
- **PHILLIPS Achieva**: 3 T scanner. Images in the SAVE set are 480x480.

2.4 Machine Learning

Our segmentation methodology will be based on Convolution Neural Networks (CNN), as in the previous projects and in line with the last trends in the scientific community [20, 21, 22, 23, 24, 25, 26]. We set to try several variations to achieve better results. In this section, we are going to show concepts related to Neural Networks and some new techniques that could aid in the improvement of results [27].

The algorithmic structures of Artificial Neural Networks (ANN) allow models that are composed of multiple layers of processing to learn data representations with multiple levels of abstraction. This set of layers composed of Neurons performs a series of linear and non-linear transformations to the input data to generate an output close to the expected (label).

Supervised learning, in this case, consists in obtaining the parameters of these transformations: the w_i (weights) and the b (biases); with the hope that these transformations produce an output that differs as little as possible to the expected output. Neural networks use the error, or loss, of their outputs in a process called *Back-propagation*, whereby these weights and biases are updated.

$$y = i \sum W_i x_i + b \qquad \text{(Neurons)}$$

The neurons form layers and multiple layers form networks as 2.4 [13].

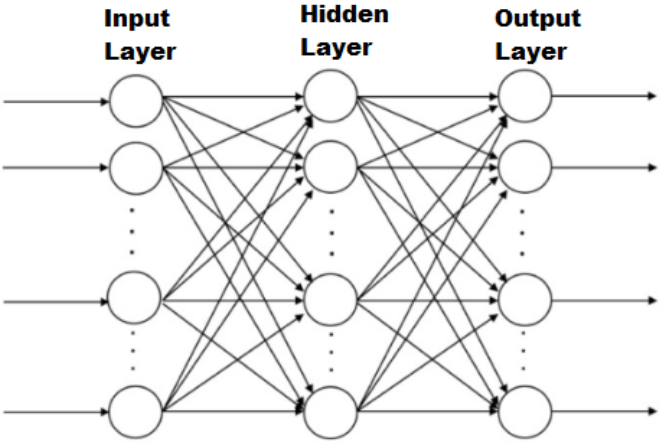

FIGURE 2.4: Neural Network

2.4.1 Neural Network hyper-parameters

Neural networks have a lot of parameters that need to be adjusted in order to obtain better results. In the following sections we will cover the most important ones.

2.4.1.1 Loss

We will use a loss function to estimate the error and to compare and measure the performance of our prediction with the respect to the expected result. Ideally, we want our cost to be zero, that is, without divergence between the estimated and expected values. Therefore, as the model is being trained, the weights of the interconnections of the neurons will gradually be adjusted until more accurate (towards zero cost) predictions are obtained.

The choice of the best function of loss resides in understanding what type of error is or is not acceptable for the problem in question.

In the particular case of applying NN to images, the *DICE* score is among the most commonly used, because it quantifies how closely the networks' outputs match the training dataset comprising hand-annotated ground truth segmentation DICE Score formula.

$$DICEScore = \frac{TP}{P + FP} \qquad \text{(DICE Score formula)}$$

Where TP is the number of True Positive pixels, P is the number of positive pixels and FP is the number of False Positive pixels, as illustrated in 2.5.

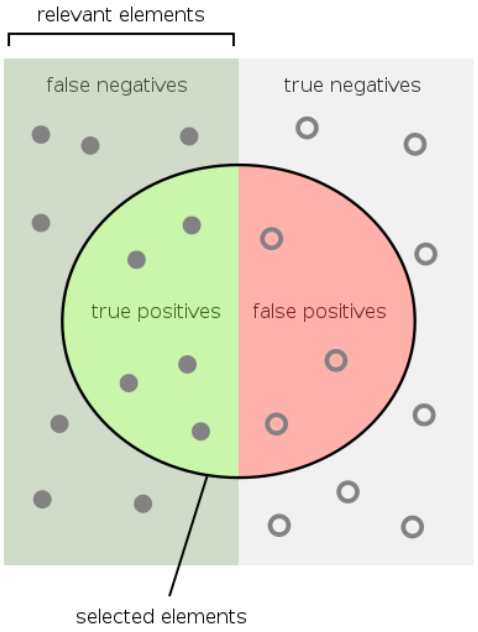

FIGURE 2.5: Classification Chart

2.4.1.2 Activation Function

Each neuron has an activation function that defines the output of the neuron. The activation function is used to introduce non-linearity in the modeling capabilities of the network. We have several options for activation functions.

The choice of a specific activation function depends on several factors: (a) the kind of data that are being processed, (b) the kind of prediction that we seek to obtain, (c) which part of the network we are in, and (d) in which kind of layer.

Most common activation functions are: *Linear*, *Sigmoid*, *Tanh*, *ReLU* and *Softmax* [13].

2.4.1.3 Optimizer

We can see the learning process as a global optimization problem where the parameters (weights and biases) must be adjusted in such a way that the loss function is minimized.

There are different optimizers that can be used: *Stochastic Gradient Descent*, *RMSprop*, *Adagrad*, *Adadelta*, *Adam*, *Adamax* and *Nadam* and many more [13].

2.4.1.4 Learning Rate

To update the weights and biases of the network, the learning rate parameters adjust how much these components change per epoch, being α in the Gradient Descent equation:

$$W_{ij} = W_{ij} - \alpha \frac{dError}{dW_{ij}} \qquad \text{(Gradient Descent equation)}$$

The learning rate decay (α) is used to decrease the learning rate as epochs go by to allow the learning to advance faster at the beginning than towards the end ("fine-tuning"). As progress is made, smaller and smaller adjustments are made to facilitate the convergence of the training process to the minimum of the loss function.

2.4.1.5 Epochs

Epochs tells us the number of times all the training data have passed through the neural network in the training process. The number of epochs is usually increased until the accuracy metric with the validation data starts to decrease, even when the accuracy of the training data continues to increase (this is when we could detect a potential overfitting).

2.4.1.6 Batch Size

We can partition the training data in smaller batches to pass them through the network. The optimal size will depend on many factors, including the memory capacity of the computer that we use to do the calculations.

2.4.1.7 Dropout

Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks.

The dropout rate estimates the provability that this event occurs.

2.4.2 Convolutional Neural Networks

A Convolutional Neuronal Network is a particular type of neural network. Already used at the end of the 90s, in the recent years, CNNs have become enormously popular while achieving very impressive results in the recognition of images, deeply impacting the area of computer vision and, accordingly, the automatic MRI cardiac multi-structures segmentation [28].

A differential feature of the CNNs is that they make the explicit assumption that the entries are images, allowing the encoding of certain properties in the architecture to recognize specific elements in the images.

In our case this network extracts features from the CMR images such as myocardium, right ventricle, fat, etc. Using these learned features, it is hypothesized that the trained model will be able to segment the images.

However this kind of features are quite complex, requiring that we first extract simpler features such as lines and curves, then circles and textures and finally the different parts of the heart. For more complex features we will need to make the network deeper as seen in figure 2.6.

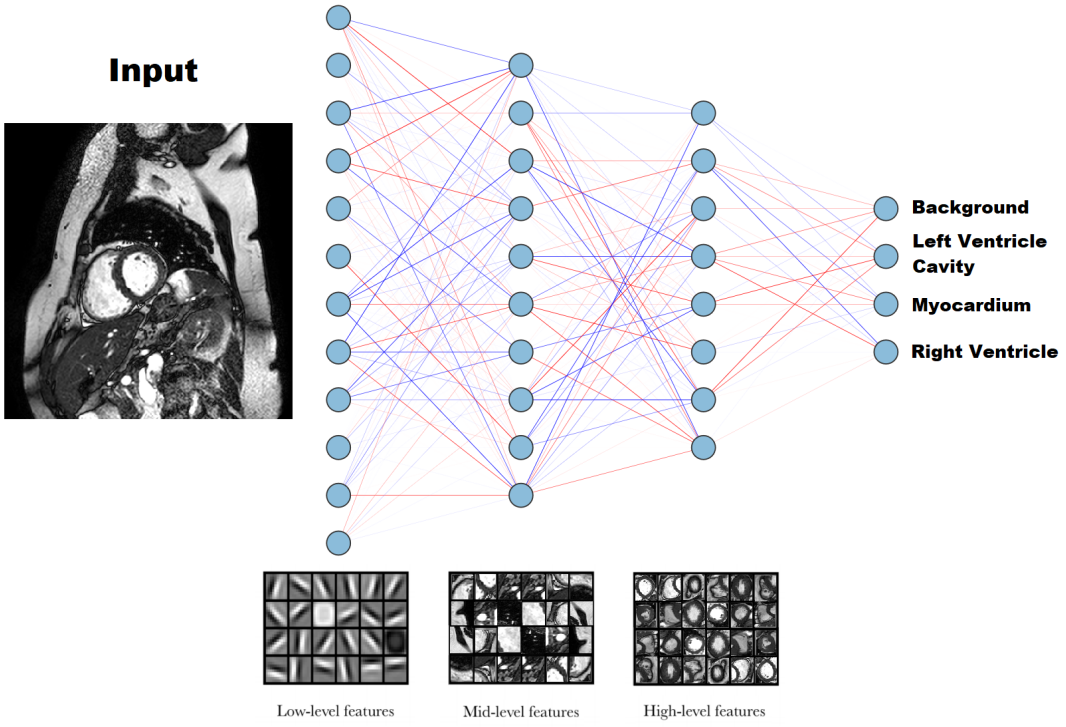

FIGURE 2.6: CNN for Cardiac MRI

In the following sections we are going to introduce the different layers of a CNN.

2.4.2.1 Convolution Layer

The main purpose of a convolutional layer 2.7 is to detect features or visual features in images such as edges, lines, color drops, etc. This is a very interesting property because, once it has learned a characteristic at a specific point in the image, it can recognize it later in any part of it.

Another important feature is that convolutional layers can learn spatial hierarchies of patterns by preserving spatial relationships. For example, a first convolutional layer can learn basic elements such as edges, and a second convolutional layer can learn patterns composed of basic elements learned in the previous layer. And so on until it learns very complex patterns. This allows convolutional neural networks to efficiently learn increasingly complex and abstract visual concepts.

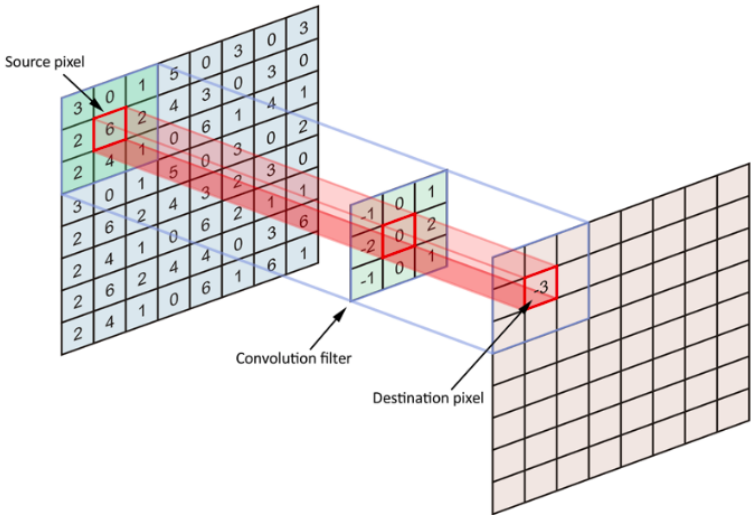

FIGURE 2.7: Convolutional Layer

There are a few hyper-parameters to consider while using convolutional layers:

- **Filter size**: Amount of neighbouring pixels, usually 3x3, 5x5 or 7x7.
- **Channel size**: Number of filters equating to number of "characteristics" that we wish to have, usually 32 or 64.

- **Padding**: Sometimes an output image of the same dimensions as the input is desired, requiring the addition of zeros, before any subsequent windowing. For this, we can use the hyper-parameter padding in the convolutional layers.
- **Stride**: Indicates the number of steps in which the filter window moves. Large stride values decrease the size of the information that will be passed to the next layer. Usually this is set to 1.

2.4.2.2 Pooling layer

Pooling layers 2.8 simplify the information collected by the convolutional layer and create a condensed version of the information contained in them.

There are several ways to condense the information. A typical choice is known as *max-pooling*, keeps the maximum value of those that were in the input *window*.

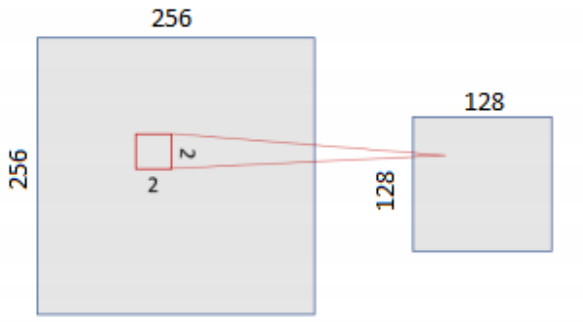

FIGURE 2.8: Pooling Layer

Average-pooling can also be used instead of max-pooling, where each group of entry points is transformed into the average value of the group of points instead of its maximum value. However, max-pooling tends to work better than alternative solutions [13].

The convolutional layer hosts more than one channel and, therefore, as we apply the max-pooling to each of them separately, the pooling layer will contain as many pooling filters as there are convolutional channels.

2.4.2.3 Fully connected layer

This layer. which is by definition densely connected, will serve to feed the final *softmax* activation function. This layer interconnects all the outputs from previous layers in order to mix all the features together to determine the output of the whole network 2.9.

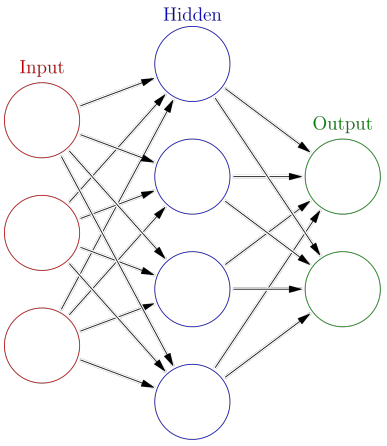

FIGURE 2.9: Fully Connected Layer

2.4.3 Metrics

In order to adjust the hyper-parameters we need some metrics for measuring the performance, and also a tool for managing and plotting them. In our case we used the loss function, the accuracy for each segment and the combined accuracy of multiple segments. We will discuss how this combinations are weighted in chapter 3.

2.4.3.1 TensorBoard

TensorBoard is a powerful tool included in Google's *TensorFlow* package that plots the desired metrics effortlessly as a web application.

With *TensorBoard*, we can check our scalar metrics, current performance, and a graph view of the current architecture 2.10. We show in blue the training performance and, in red, the validation performance.

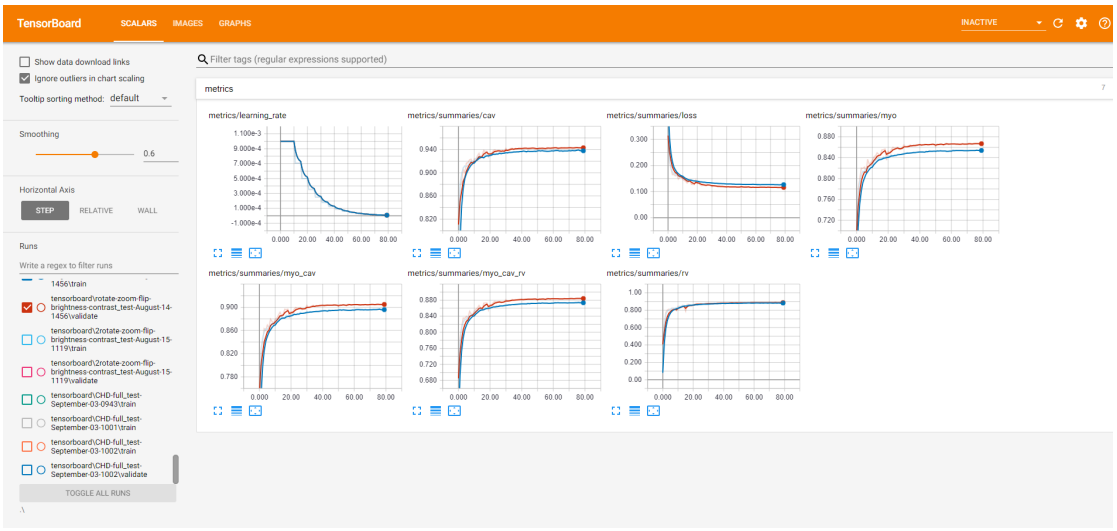

FIGURE 2.10: TensorBoard

2.4.4 Useful variations and combinations

plentiful literature exists to improve neural networks depending on the purpose. In the following section, we introduce a few techniques expected to improve performance in this environment.

2.4.4.1 Spatial Pyramid Pooling

The Spatial Pyramid Pooling [16] is a layer that can be added before the fully connected layer, allowing the network to be trained and predict over images of multiple sizes. Consequently improving the performance because there is no information lost while down sampling, cropping or warping. In 2.11 a comparison is shown between a model altering the images versus a SPP model.

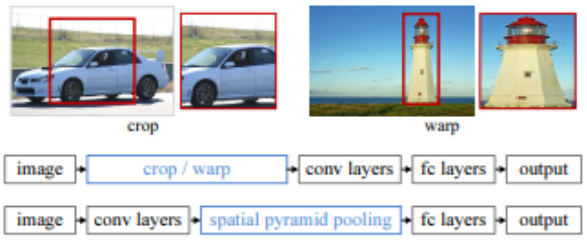

FIGURE 2.11: Spatial Pyramid Pooling

This kind of layer could be useful for our prediction in the SAVE dataset since it contains images bigger than 256x256, the size that our networks are able to work with.

2.4.4.2 Data Augmentation

Overfitting is among the most common problems when working with NNs, data augmentation helps reduce this [21]. Data augmentation is a training method for our network, it increases the amount of data available by altering the current images of the dataset to obtain new ones.

In our particular case, learning multiple variants from our features could help our network. For example, we could augment our data by rotating or scaling (zooming in) some images. Indeed, the most common augmentation methods are: rotation, flipping, zooming, brightness equalization and contrast equalization.

2.4.4.3 Learning Transfer

Learning Transfer methods focus on storing knowledge gained while solving one problem and applying it to a different but somewhat related problem, using features obtained by training a network as initial features for a second network.

Transfer learning allows the learned model on the "source task" to be adapted to a different, but related, "target task". In our case, it could enable a model that learns LV contours (source) to train another network to estimate RV contours (target). According to [21, 29], transferring features even from distant tasks can be better than using random features, as high-level information is, in theory, preserved.

Also, the source and target datasets do not need to be from the same distribution, which could be useful for processing datasets from different scanners (multi-site).

2.4.4.4 Network combination

Since we are trying to segment images in multiple regions, our *DICE* score was a weighted combination of the *DICE* score for each segment according to the ground truth. This may however, induce a lack of learning in some regions.

Network combination splits the training and prediction in multiple networks that specialize in one or more segments, and then aggregates the different outputs.

Training and predicting each region separately, or different combinations thereof, can improve the results. However this comes at the risk of overfitting and at the expense of higher computational time. We test and analyzed the use of this technique in our network.

Chapter 3

Setup

This chapter shows how we use the concepts introduced in the previous chapter to build our program. We discuss the procedures and singularities used, and the solutions that we proposed to sort our constrains.

The whole project can be seen in four parts, shown in 3.1 representing our executables. We explain what these parts do, as well as their inputs and outputs, in the following sections.

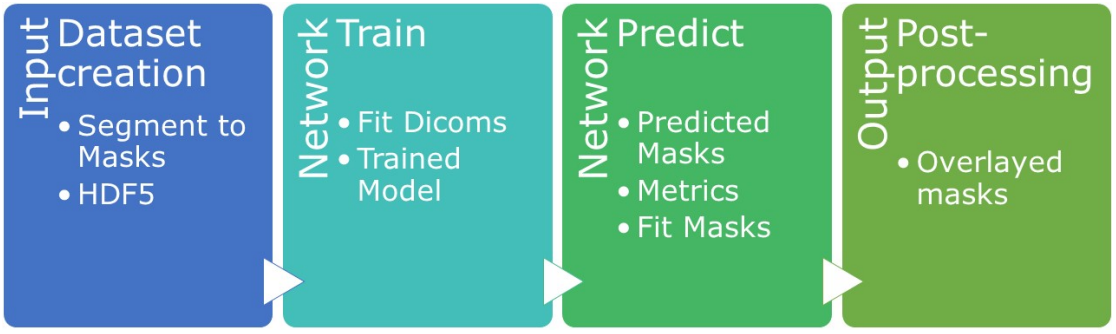

FIGURE 3.1: Project parts

Except the contours to ground truth part, everything is coded using *Python* following the structure displayed in Figure 3.2.

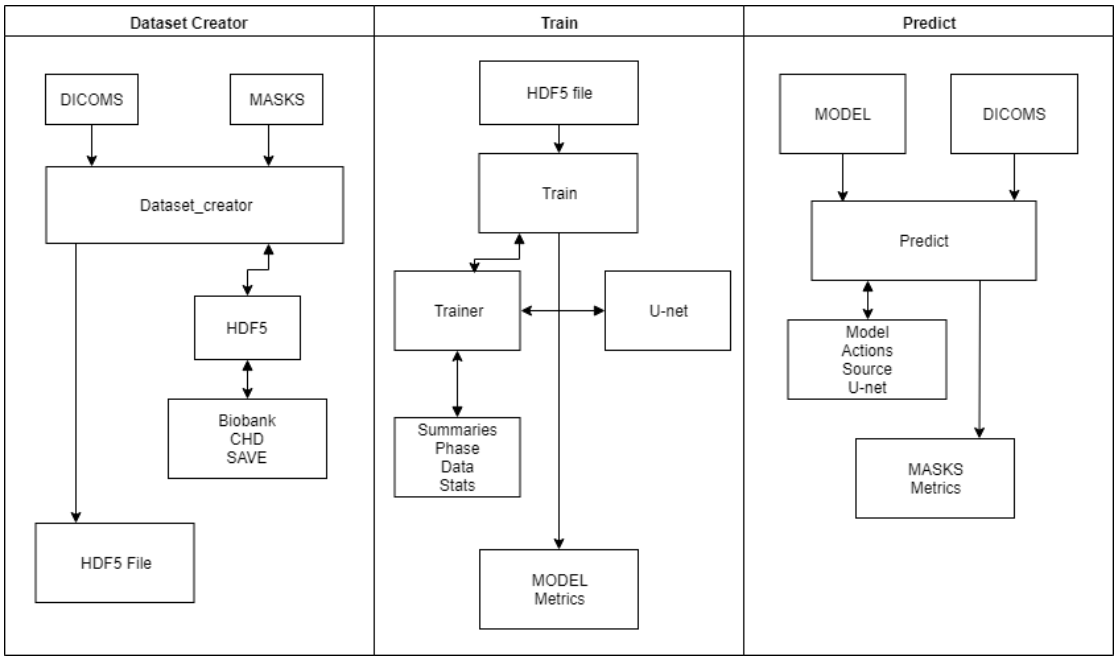

FIGURE 3.2: Python Code Diagram

Later on, in the experiments and results chapters we will introduce a few changes to test our proposals for improving the sandbox approach performance.

3.1 Data

Our setup is tailored to fit the dataset structures, in the following sections we are going to show these structures and how we deal with them to feed the neural network.

There are three different data sources:

- **United Kingdom Biobank**: dataset of 83,725 short axis CMR images. DICOM images and ground truth presented in mirrored folders.
- **Cardiac Heart Disease**: dataset of 105 cases, its ground truth comes from expert analysts in the Department.
- **SAVE**: big dataset from which we used 32 randomly selected cases (8 cases per scanner). The analyst segmented them for us.

3.2 Input Pipeline

Before training our network data must be arranged. Masks were obtained from the contours which the analyst manually traced from the DICOM images. Once we had them, we were able to generate our *HDF5* file with the masks and the DICOM images (Figure 3.3).

- **DICOM** is the international standard to transmit, store, retrieve, print, process, and display medical imaging information [30].
- **Mask** is a *png* file that displays in different grey scale levels the different regions in a CMR image.
- **HDF5** is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data [31].

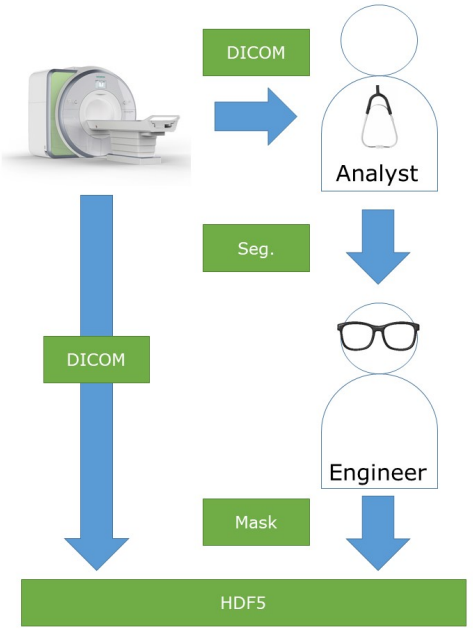

FIGURE 3.3: Input Pipeline

3.2.1 Contours to Ground Truth Masks

Ground truth comes as contours from the analysts, typically radiology experts. We require the ground truth as *png* files, or masks, therefore we needed to process this contours firstly. We can see and example of DICOM and mask in the figure 3.4.

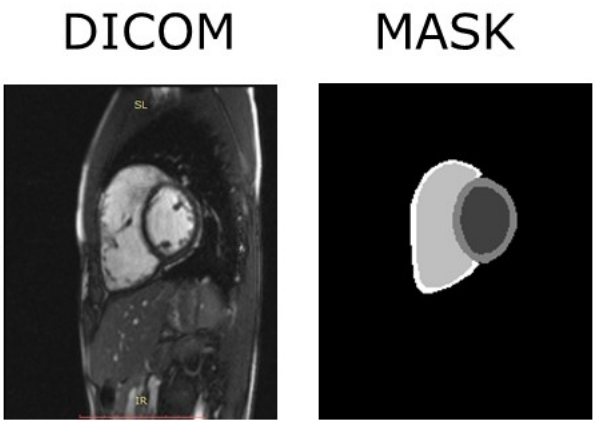

FIGURE 3.4: DICOM and Mask

The contours come as *segments* which are *struct* files containing 80 image coordinates, defining the contour for both End Diastole and End Sistole states. We used *Matlab* to convert these contours to masks, using morphological operations.

In the masks there are the following regions: right ventricular wall, right ventricular cavity, left ventricular cavity and myocardium.

Because contours were traced at a higher resolution, when obtaining the masks we encountered overlapping in some of the regions and sometimes the outer regions were not closed. We dealt with these problems while creating the masks by using morphological operations.

The *Matlab* coding uses a main function that feeds both ED and ES mask generators while iterating trough the cases and slices, following the diagram in the figure 3.5.

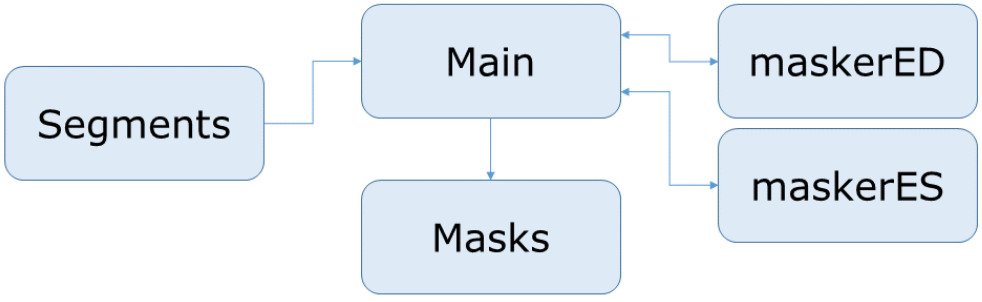

FIGURE 3.5: Matlab Code Diagram

3.2.2 Dataset Creator

As shown in the figure 3.2, the dataset creator generates a *HDF5* file appending the DICOMs and masks, with a *HDF5* function that calls the convenient function (CHD, UKB or SAVE) to understand the architecture of the dataset.

In this stage we also prepared the images to fit into the network, resizing them to 256x256. Bigger images were down-sampled and smaller images were up-sampled, using nearest-neighbour interpolation for the masks and bi-cubic interpolation for DICOMs, and then padded with zeros when needed.

3.3 Neural Network

In the following section we present the architecture of our network as well as how we managed it and its hyper-parameters.

3.3.1 U-net

We used the *U-net* architecture [1] shown in the figure 3.6. It is composed by convolutional layers with 16 3x3 filters with stride equal to 2, non padded and activated by *ReLU*. After each convolution layer there is a 2x2 max-pooling layer in order to reduce the amount of features.

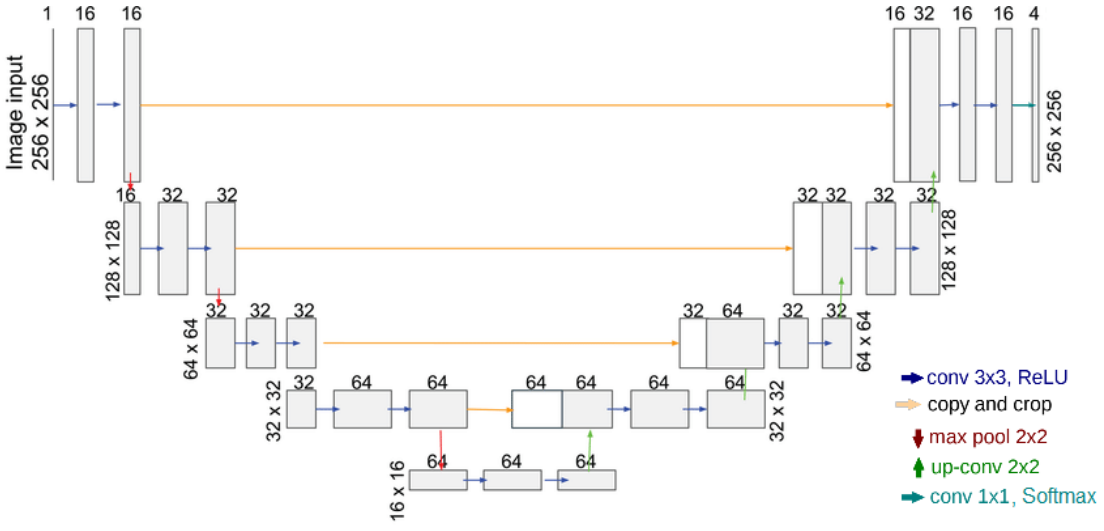

FIGURE 3.6: U-net

3.3.2 Model

We call Model the functions that manage the network, it means that these functions are the ones that train the network to obtain the optimal weights, called Saved Model. It also manages the prediction or the process to obtain Masks using a Saved Model and raw DICOMs (Figure 3.7).

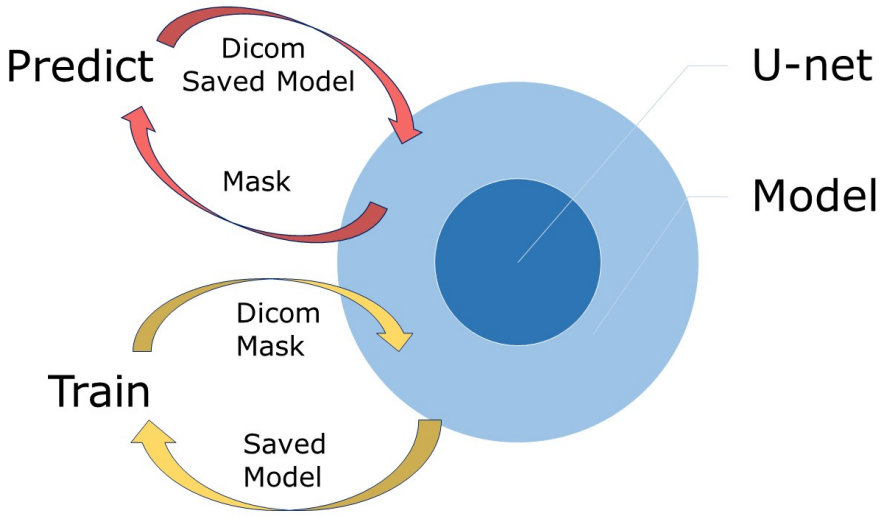

FIGURE 3.7: Model Usage

3.3.3 Training

For training we need a dataset including masks and DICOMs, already fitted and appended to a *HDF5* file. As output we will receive a saved model and the performance metrics for this process. The best way to look at them and compare the different Saved Models is to use *TensorBoard*.

As starting learning rate we used $1e^{-3}$ and as optimizer we used *SGD* with *DICE* score as loss function, weighted as Weighted DICE score. The number of epochs was set to 80 and the dropout rate to 0.8. We use 256 as number of training batches to run before performing validation and 32 as batch size.

$$DICE = 0.7 \times DICE_{MYO} + 0.2 \times DICE_{LV} + 0.1 \times DICE_{RV} \quad \text{(Weighted DICE score)}$$

3.3.4 Prediction

During prediction, masks are automatically generated using DICOMs and a saved model. It is important to notice that in this step the re-sizing of the DICOMs is done while prediction, not when creating the dataset.

After predicting the masks we also need to fit them to the original size to match their original DICOMs. We fit them using the procedure used when creating the datasets.

3.4 Post-processing

To visually check the quality of the predictions, especially when we did not have the ground truth for the test set, we compute the DICOMs and masks in the output of the prediction step and convert them to an *alpha* color space, then we are able to fade the mask over the DICOM as seen in 3.8.

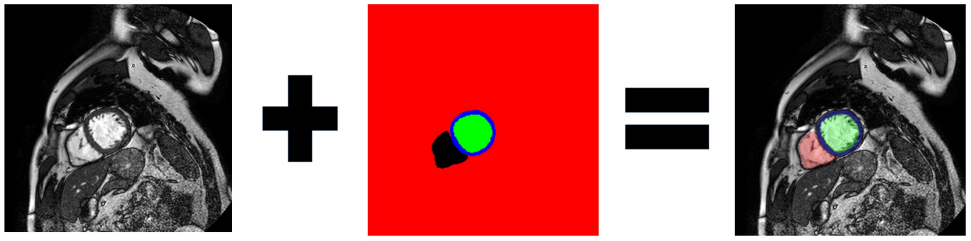

FIGURE 3.8: Overlayed Output

Chapter 4

Experiments

With the aim of tuning our network and try some modifications, we proposed some tests, some of them have been evaluated visually and some of them have also been evaluated with metrics (as we obtained the ground truth for the test set in the third part of the project).

Through these tests and their results we will obtain the best model, considering the computational and data constrains.

4.1 Sandbox approach

In this first test we saw how the network behaves when using a simple model trained with the UK Biobank dataset. The setup used is the one described in Chapter 3.

The sandbox approach is to use the most simple model able to be trained with a single dataset and able to predict over the targeted test set, both GT mask and predicted mask have 4 channels. It means that this model is able to predict over multiple scanners' datasets where images can be bigger or smaller than 256x256.

We compared the results of this test to the other tests results to see how good the improvements are.

4.2 Data Augmentation test

In this test we used 5 different data augmentation methods to prevented overfitting while training and also use a more heterogeneous images, enabling the network to learn a wider range of features. These methods are: rotation, zooming, flipping, contrast equalization and brightness equalization. The chances for them to be used are random and so are their parameters (angle, amount of zooming and level of equalization). Each input image went through all these methods before feeding the network.

In this test we firstly evaluated which augmentations should be used and then computed which is the impact of them.

4.3 Dice score weighting

As our loss function has a strong impact in the features we learn and we balance it with the weighted DICE score, we decide to test which is the best weighting and also analyze the performance for each segment when changing the weights.

For this test we used 6 different weightings and referred to them with a number ABC, where A is the weighting for myocardium, B is the weighting for LV cavity and C is the weighting for RV cavity.

4.4 SPP layer test

As we were losing information when down-sampling images bigger than 256x256, for which the literature suggested an SPP layer. This test was designed to assess its performance within the network.

4.5 Filter channel size test

This test measures if we have enough data and computational power to set the hyper-parameter to 32, and how much this affects the performance.

Chapter 5

Results and Discussion

5.1 Results

In the following chapter we show and discuss the numeric results for all tests, plotting their DICE score mean and median (Y axis) for each scanner and for each segment (X axis).

5.1.1 Sandbox approach

The results for our sandbox approach (found in A.1) are plotted in figure 5.1. As we can see, we are just able to predict over Siemens Aera and the other scanners' performance are poor or null.

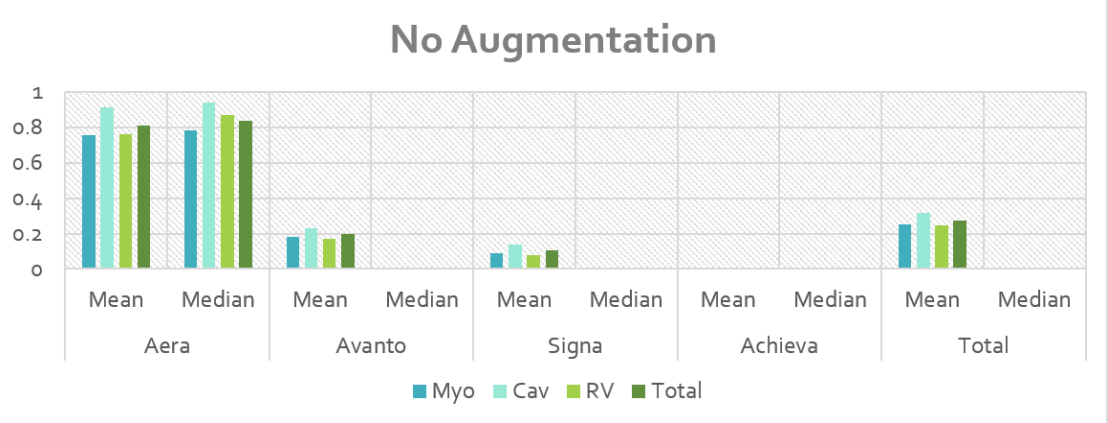

FIGURE 5.1: No Augmentation Model Results

5.1.2 Data augmentation test

After creating a model using data augmentation, the results are very promising as we can see in figure 5.2 using data in table A.2. We can see how data augmentation enables the prediction over other scanners, and even achieving the best result on the Phillips Achieva.

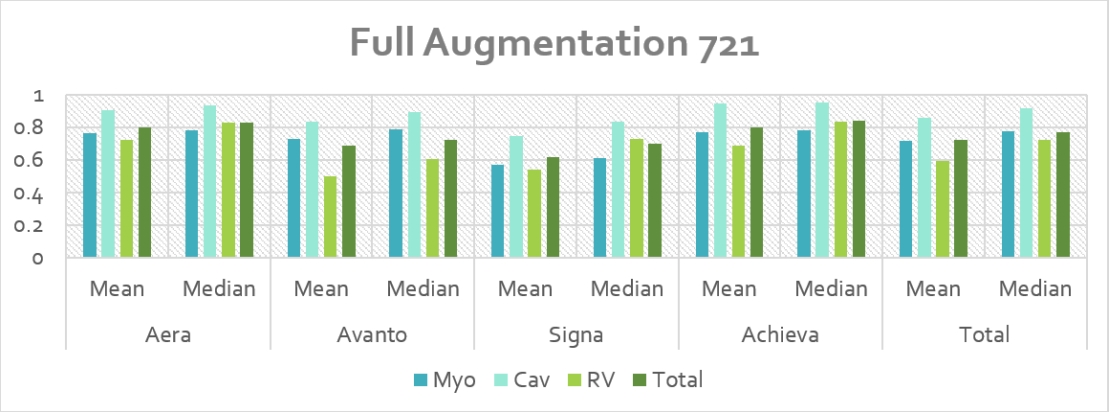

FIGURE 5.2: Full Augmentation Model Results

5.1.3 Training with multiple datasets test

By training the models with multiple datasets, the results when predicting are plotted in figure 5.3 using data in table A.3. We can confirm that using multiple datasets helps improve performance over all scanners by about 6%.

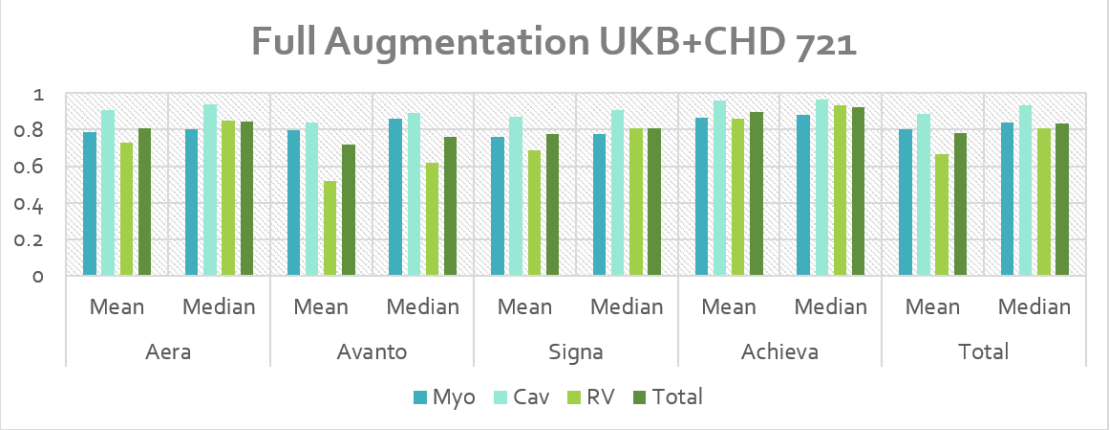

FIGURE 5.3: Full Augmentation with UKB+CHD Model Results

5.1.4 DICE score weighting test

In this test we obtained a lot of results, displayed in the tables A.4 for 721, A.5 for 217, A.7 for 424, A.8 for 613, A.9 for 622 and A.6 for 523.

The best results were obtained on 523, plotted in figure 5.4. We can see a minor overall improvement of 3% mostly due to improvements in the right ventricle segmentation prediction.

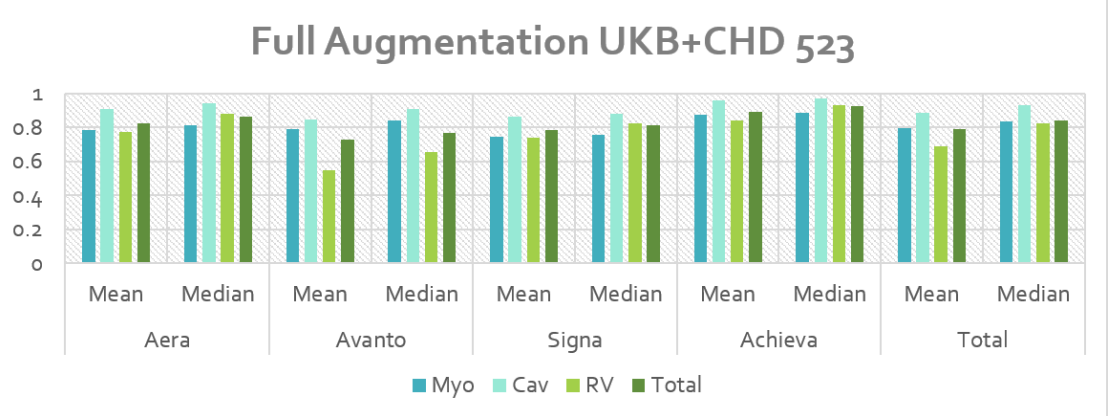

FIGURE 5.4: Full Augmentation with UKB+CHD 523 weighted Model Results

5.2 Scanners Recap and Discussion

We can see that the performance over Siemens Aera (shown in A.10) has been stable, even performing a little bit better over the latest models, as we can see in figure 5.5.

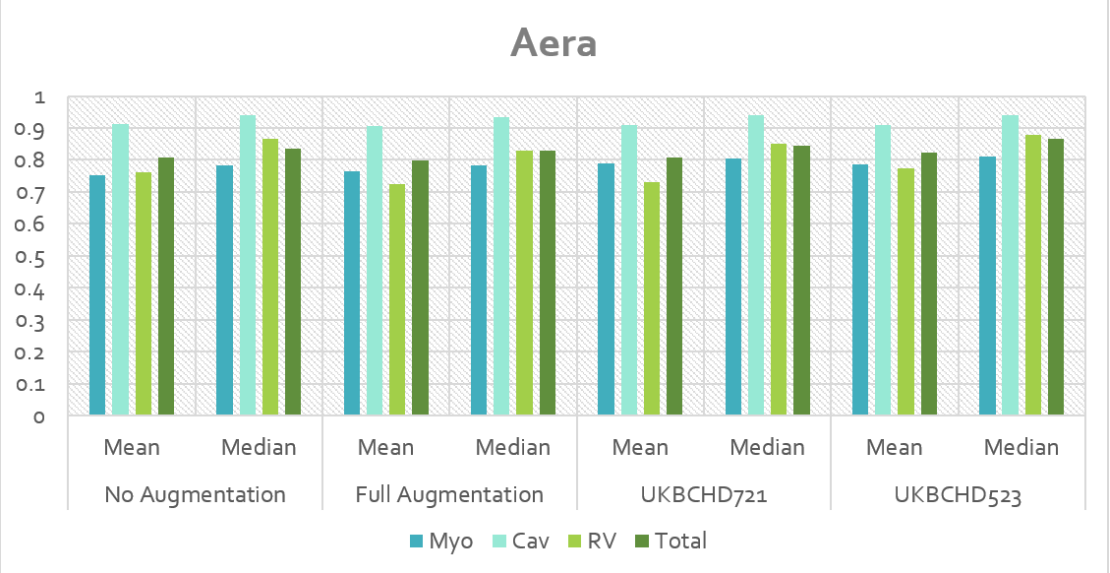

FIGURE 5.5: Full Augmentation with UKB+CHD 523 weighted Model on Siemens Aera

Siemens Avanto, has achieved the worst results but has improved its performance, specially after data augmentation. The results are plotted in figure 5.6 using data in table A.11.

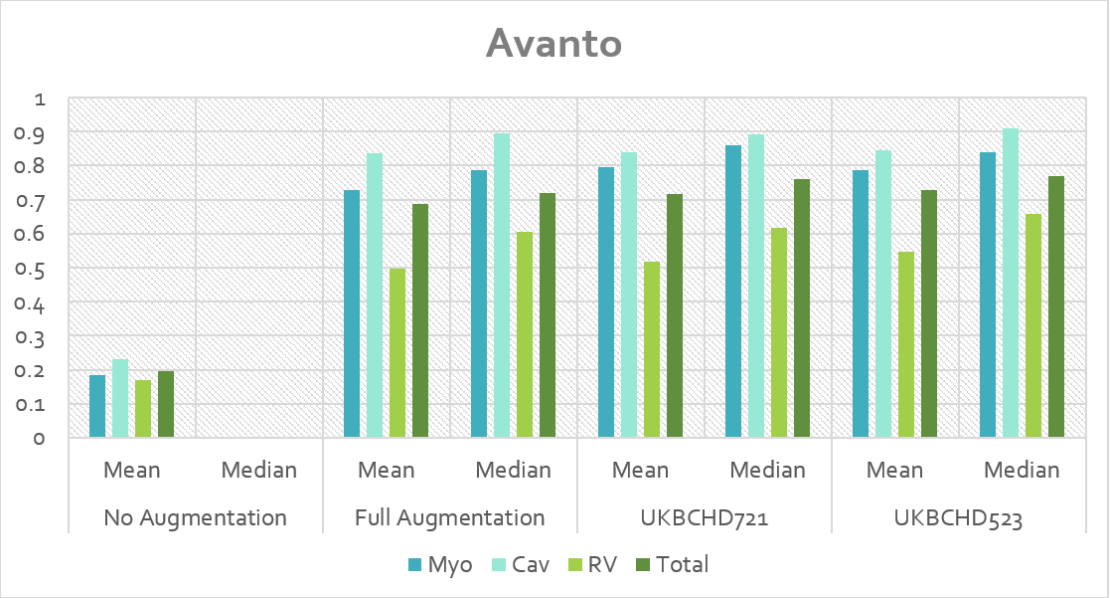

FIGURE 5.6: Full Augmentation with UKB+CHD 523 weighted Model on Siemens Avanto

GE Signa had slightly better results than Siemens Avanto, but similar performance. We can see its results in figure 5.7 using data in table A.12.

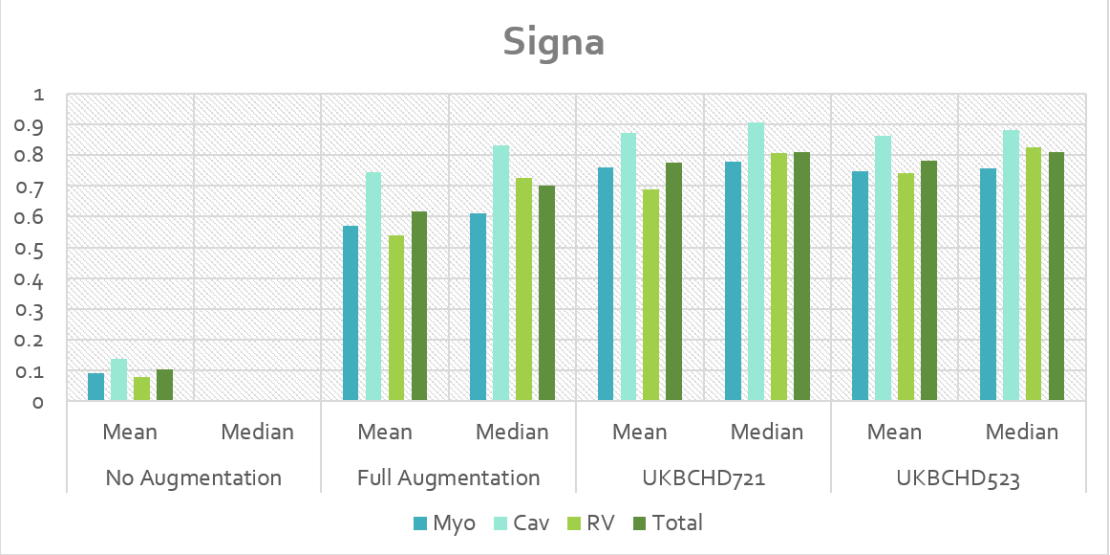

FIGURE 5.7: Full Augmentation with UKB+CHD 523 weighted Model on GE Signa HDxt

Predictions over Phillips Achieva have been the most successful and promising ones because, after having 0 images predicted at the beginning, it achieved the best overall results at the end. We can see the results in figure 5.8 using the data in table A.13.

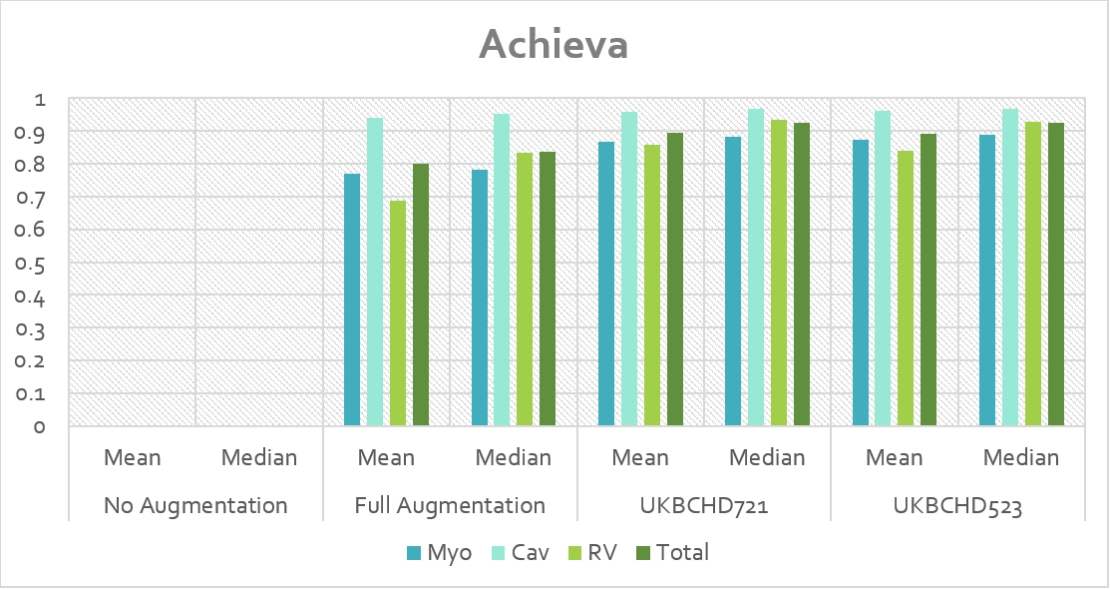

FIGURE 5.8: Full Augmentation with UKB+CHD 523 weighted Model on Phillips Achieva

5.3 SPP layer

While running this test we saw that there were no improvements by adding this layer. Surprised we tried to figure out why our test were not confirming the literature and found that the images that are bigger than 256x256 were up-sampled to 480x480 during image acquisition for an unknown reason. That confirmed that there was no information lost in the process.

5.4 Filter channel size

We prepared a network with a filter channel size of 32 and tried to run it, unfortunately there was not enough memory in our machine and therefore the test crashed.

Chapter 6

Conclusions

First of all, the data augmentation test has proven that by applying this technique we are able to predict over new scanners without having any images from them in our training or validation set. Moreover, we still predict equally well over the training set scanner.

Our explanation to this phenomenon is that by making the set more heterogeneous, we unlock some learning over features that were missed when not augmenting. In other words, we allow the network to choose from a wider range of features.

Secondly, the dataset combination that we made when adding the CHD dataset to the training set was quite successful, because it was just representing 3% of the final training set and it increased the global performance by 6%. So we conclude that adding small specific sets, that could make the original set more heterogeneous, can worthily improve prediction performance.

According to our DICE score weighting results, we conclude that 523 weighting is the the optimal for our purpose even though the performance has just increased slightly over 1%.

An interesting observation was that changing the weights in the myocardium had a strong effect on the performance over the cavity.

After looking for an explanation we concluded that finding the myocardium is really close to finding the cavity, because the second one is just the region inside the first one.

While this conclusion may sound useless, we can use this result to improve our performance on the right ventricle cavity synthetically.

As we do not have ground truth for right ventricle wall we could create it artificially using the right ventricle cavity and some simple morphological operations. By doing that we may learn the features related to right ventricle wall and increase our performance over the right ventricle cavity. Hence we thing that it could be really interesting to test this method in future work.

Bearing in mind the computational restrictions and data limitations we faced, I am satisfied with the improvements and results, which in some cases are close to those of the expert analysts. However further work is required to achieve that level of unsupervised expertise, as in some cases, the performance for certain images is still far from acceptable.

Appendix A

Result's tables

A.1 First Approach

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.752	0.783	0.183	0	0.091	0	0	0	0.253	0
Cav	0.911	0.94	0.232	0	0.139	0	0	0	0.315	0
RV	0.762	0.866	0.171	0	0.079	0	0	0	0.248	0
Total	0.808	0.836	0.196	0	0.103	0	0	0	0.272	0

Table A.1: First Approach

A.2 Data Augmentation

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.765	0.783	0.728	0.786	0.571	0.611	0.77	0.781	0.718	0.775
Cav	0.906	0.933	0.836	0.894	0.745	0.831	0.942	0.952	0.858	0.916
RV	0.723	0.828	0.496	0.606	0.539	0.725	0.688	0.834	0.592	0.722
Total	0.798	0.829	0.687	0.719	0.618	0.701	0.8	0.838	0.723	0.77

Table A.2: Data augmentation

A.3 UK Biobank + CHD

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.788	0.805	0.796	0.861	0.761	0.777	0.867	0.883	0.803	0.84
Cav	0.909	0.939	0.838	0.893	0.872	0.905	0.96	0.967	0.885	0.935
RV	0.729	0.85	0.518	0.618	0.688	0.807	0.858	0.933	0.664	0.809
Total	0.808	0.845	0.717	0.759	0.774	0.808	0.895	0.925	0.784	0.832

Table A.3: UK Biobank + CHD

A.4 DICE Score Weighting

A.4.1 0.7 Myocardium + 0.2 Cavity + 0.1 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.788	0.805	0.796	0.861	0.761	0.777	0.867	0.883	0.803	0.84
Cav	0.909	0.939	0.838	0.893	0.872	0.905	0.96	0.967	0.885	0.935
RV	0.729	0.85	0.518	0.618	0.688	0.807	0.858	0.933	0.664	0.809
Total	0.808	0.845	0.717	0.759	0.774	0.808	0.895	0.925	0.784	0.832

Table A.4: DICE: 0.7 Myocardium + 0.2 Cavity + 0.1 Right Ventricle

A.4.2 0.2 Myocardium + 0.1 Cavity + 0.7 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.77	0.799	0.762	0.811	0.737	0.737	0.807	0.827	0.769	0.808
Cav	0.902	0.934	0.839	0.9	0.88	0.9	0.94	0.956	0.881	0.926
RV	0.754	0.858	0.56	0.666	0.731	0.822	0.8	0.907	0.682	0.822
Total	0.808	0.862	0.721	0.761	0.783	0.82	0.849	0.894	0.777	0.824

Table A.5: DICE: 0.2 Myocardium + 0.1 Cavity + 0.7 Right Ventricle

A.4.3 0.5 Myocardium + 0.2 Cavity + 0.3 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.785	0.81	0.787	0.84	0.747	0.758	0.872	0.888	0.797	0.833
Cav	0.91	0.939	0.846	0.909	0.863	0.881	0.961	0.969	0.887	0.929
RV	0.773	0.878	0.546	0.657	0.74	0.825	0.839	0.929	0.69	0.826
Total	0.823	0.865	0.727	0.768	0.783	0.811	0.891	0.925	0.791	0.839

Table A.6: DICE: 0.5 Myocardium + 0.2 Cavity + 0.3 Right Ventricle

A.4.4 0.4 Myocardium + 0.2 Cavity + 0.4 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.773	0.795	0.785	0.833	0.745	0.749	0.847	0.867	0.789	0.821
Cav	0.907	0.94	0.846	0.916	0.879	0.887	0.956	0.964	0.888	0.929
RV	0.762	0.854	0.532	0.656	0.713	0.809	0.796	0.915	0.668	0.796
Total	0.814	0.856	0.721	0.751	0.779	0.811	0.866	0.912	0.781	0.825

Table A.7: DICE: 0.4 Myocardium + 0.2 Cavity + 0.4 Right Ventricle

A.4.5 0.6 Myocardium + 0.1 Cavity + 0.3 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.778	0.792	0.785	0.848	0.729	0.729	0.87	0.885	0.792	0.834
Cav	0.908	0.941	0.844	0.906	0.844	0.883	0.958	0.967	0.882	0.927
RV	0.768	0.869	0.556	0.672	0.681	0.801	0.833	0.931	0.681	0.818
Total	0.818	0.862	0.729	0.766	0.752	0.798	0.887	0.924	0.785	0.831

Table A.8: DICE: 0.6 Myocardium + 0.1 Cavity + 0.3 Right Ventricle

A.4.6 0.6 Myocardium + 0.2 Cavity + 0.2 Right Ventricle

	Aera		**Avanto**		**Signa**		**Achieva**		**Total**	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Myo	0.788	0.814	0.803	0.852	0.745	0.762	0.87	0.884	0.804	0.843
Cav	0.909	0.939	0.848	0.91	0.864	0.879	0.96	0.969	0.887	0.931
RV	0.76	0.857	0.478	0.612	0.67	0.809	0.834	0.934	0.646	0.801
Total	0.819	0.855	0.709	0.734	0.759	0.807	0.888	0.927	0.78	0.83

Table A.9: DICE: 0.6 Myocardium + 0.2 Cavity + 0.2 Right Ventricle

A.5 Scanners

A.5.1 Siemens Aera

	No Augmentation		**Full Augmentation**		**UKBCHD721**		**UKBCHD523**	
	Mean	Median	Mean	Augmentation	Mean	Median	Mean	Median
Myo	0.752	0.783	0.765	0.783	0.788	0.805	0.785	0.81
Cav	0.911	0.94	0.906	0.933	0.909	0.939	0.91	0.939
RV	0.762	0.866	0.723	0.828	0.729	0.85	0.773	0.878
Total	0.808	0.836	0.798	0.829	0.808	0.845	0.823	0.865

Table A.10: Siemens Aera

A.5.2 Siemens Avanto

	No Augmentation		**Full Augmentation**		**UKBCHD721**		**UKBCHD523**	
	Mean	Median	Mean	Augmentation	Mean	Median	Mean	Median
Myo	0.183	0	0.728	0.786	0.796	0.861	0.787	0.84
Cav	0.232	0	0.836	0.894	0.838	0.893	0.846	0.909
RV	0.171	0	0.496	0.606	0.518	0.618	0.546	0.657
Total	0.196	0	0.687	0.719	0.717	0.759	0.727	0.768

Table A.11: Siemens Avanto

A.5.3 GE Signa HDxt

	No Augmentation		**Full Augmentation**		**UKBCHD721**		**UKBCHD523**	
	Mean	Median	Mean	Augmentation	Mean	Median	Mean	Median
Myo	0.091	0	0.571	0.611	0.761	0.777	0.747	0.758
Cav	0.139	0	0.745	0.831	0.872	0.905	0.863	0.881
RV	0.079	0	0.539	0.725	0.688	0.807	0.74	0.825
Total	0.103	0	0.618	0.701	0.774	0.808	0.783	0.811

Table A.12: GE Signa HDxt

A.5.4 Phillips Achieva

	No Augmentation		**Full Augmentation**		**UKBCHD721**		**UKBCHD523**	
	Mean	Median	Mean	Augmentation	Mean	Median	Mean	Median
Myo	0	0	0.77	0.781	0.867	0.883	0.872	0.888
Cav	0	0	0.942	0.952	0.96	0.967	0.961	0.969
RV	0	0	0.688	0.834	0.858	0.933	0.839	0.929
Total	0	0	0.8	0.838	0.895	0.925	0.891	0.925

Table A.13: Phillips Achieva

Appendix B

Python Packages

# Name	Version	Build
absl-py	0.2.2	<pip>
astor	0.6.2	<pip>
astroid	1.6.1	<pip>
blas	1.0	mkl
bleach	1.5.0	<pip>
ca-certificates	2018.03.07	0
certifi	2018.4.16	py36_0
colorama	0.3.9	py36h029ae33_0
cycler	0.10.0	py36h009560c_0
dicom	0.9.9.post1	<pip>
enum34	1.1.6	<pip>
freetype	2.8	h51f8f2c_1
gast	0.2.0	<pip>
grpcio	1.12.0	<pip>
h5py	2.8.0	py36h3bdd7fb_0
hdf5	1.10.2	hac2f561_1
html5lib	0.9999999	<pip>
icc_rt	2017.0.4	h97af966_0
icu	58.2	ha66f8fd_1
intel-openmp	2018.0.3	0
isort	4.3.4	<pip>

jpeg	9b	hb83a4c4_2
kiwisolver	1.0.1	py36h12c3424_0
lazy-object-proxy	1.3.1	<pip>
libiconv	1.15	h1df5818_7
libpng	1.6.34	h79bbb47_0
libtiff	4.0.9	hb8ad9f9_1
libxml2	2.9.8	hadb2253_1
libxslt	1.1.32	hf6f1972_0
lxml	4.2.2	py36hef2cd61_0
Markdown	2.6.11	<pip>
matplotlib	2.2.2	py36h153e9ff_1
mccabe	0.6.1	<pip>
mkl	2018.0.3	1
mkl_fft	1.0.1	py36h452e1ab_0
mkl_random	1.0.1	py36h9258bd6_0
numpy	1.14.5	py36h9fa60d3_0
numpy-base	1.14.5	py36h5c71026_0
olefile	0.45.1	py36_0
opencv	3.3.1	py36h20b85fd_1
openssl	1.0.2o	h8ea7d77_0
pandas	0.23.1	py36h830ac7b_0
pillow	5.1.0	py36h0738816_0
pip	10.0.1	py36_0
protobuf	3.5.2.post1	<pip>
pydicom	1.0.2	<pip>
pylint	1.8.2	<pip>
pyparsing	2.2.0	py36h785a196_1
pyqt	5.9.2	py36h1aa27d4_0
python	3.6.5	h0c2934d_0
python-dateutil	2.7.3	py36_0
pytz	2018.4	py36_0
qt	5.9.6	vc14h62aca36_0
scipy	1.1.0	py36h672f292_0
setuptools	39.2.0	py36_0

sip	4.19.8	py36h6538335_0
six	1.11.0	py36h4db2310_1
sqlite	3.24.0	h7602738_0
tb-nightly	1.9.0a20180604	<pip>
tensorboard	1.8.0	<pip>
tensorflow-gpu	1.8.0	<pip>
termcolor	1.1.0	<pip>
tk	8.6.7	hcb92d03_3
tornado	5.0.2	py36_0
typing	3.6.4	py36_0
vc	14	h0510ff6_3
vs2015_runtime	14.0.25123	3
Werkzeug	0.14.1	<pip>
wheel	0.31.1	py36_0
wincertstore	0.2	py36h7fe50ca_0
wrapt	1.10.11	<pip>
zlib	1.2.11	h8395fce_2

Appendix C

User Guide

C.1 Environment and workspace

The first thing to install is a python package manager. We used Anaconda with Python 3.7. The code should be reasonably forwards compatible with newer versions.

Then we should create a new Anaconda environment, using anaconda prompt:

```
conda create --name ML
activate ML
```

This will create a new environment named ML, and switch to it. Packages can then be added through either:

```
conda install package-name.
```

For: tensorflow, h5py, matplotlib, pillow, opencv, lxml, colorama and scipy.

```
pip install package-name.
```

For pydicom

As work space we used cardiac folder, which is inside our user folder. Here is where we synchronized the repository.

C.2 Dataset Creator

In the script create_dataset.py there are already typed all the different possibilities to add datasets. The user just needs to comment out the kind of datasets that are not going to be used and retype the path to match the sources.

C.3 Train

In example_train.py there is plenty of examples for training. The user just needs to set the source train dataset, the amount of classes to be trained (3 or 4), the name that the output model should have, the path where the model should be saved, and which augmentation methods should be applied (using true of false or any of the following augmentation methods: rotation, flip, zoom, contrast and brightness).

While training the user can track the performance of the training and its metrics by opening a anaconda prompt and typing:

```
tensorboard --logdir=model-folder
```

And then opening a browser in the direction provided.

C.4 Predict

The latest script for prediction is save2_predict.py. To use it the user just needs to type the path for the trained model and the path to the target dataset.

Resulting images will be found in a folder with the same name of the model and the script will also output histograms with model performance.

Bibliography

[1] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *arxiv* (2015). URL: https://arxiv.org/pdf/1505.04597.pdf.

[2] Thomas A. Gaziano, MSc Asaf Bitton, Shuchi Anand, Shafika Abrahams-Gessel, and Adrianna Murphy. "Growing Epidemic of Coronary Heart Disease in Low- and Middle-Income Countries". In: *NCBI* (2011). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2864143/.

[3] Microsoft. *Windows 10*. URL: https://www.microsoft.com/en-nz/windows.

[4] Anaconda. *Anaconda for windows*. URL: https://www.anaconda.com/.

[5] Microsoft. *Visual Studio Code*. URL: https://code.visualstudio.com/.

[6] Python. *Python*. URL: https://www.python.org.

[7] Mathworks. *Matlab*. URL: https://au.mathworks.com/products/matlab.html.

[8] Tensorflow. *Tensorflow*. URL: https://www.tensorflow.org/.

[9] OpenCV. *OpenCV2*. URL: https://opencv.org/.

[10] Jordi Torres. *Hello World en TensorFlow*. UPC, 2016. URL: https://torres.ai/research-teaching/tensorflow/libro-hello-world-en-tensorflow/.

[11] Itay Lieder, Tom Hope, and Yehezkel S. Resheff. *Learning TensorFlow*. O'Reilly Media, Inc., 2017. URL: https://www.safaribooksonline.com/library/view/learning-tensorflow/9781491978504/.

[12] Keras. *Keras*. URL: https://keras.io/.

[13] Jordi Torres. *Deep Learning: Introducción práctica*. UPC, 2018. URL: https://torres.ai/artificial-intelligence-content/first-contact-deep-learning-practical-introduction-keras/.

[14] Nvidia. *GTX Titan X*. URL: https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x.

[15] Jupyter. *Jupyter Notebook*. URL: http://jupyter.org/.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Rena, and Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *arxiv* (2015). URL: https://arxiv.org/pdf/1406.4729.pdf.

[17] Wikipedia. *Heart*. URL: https://en.wikipedia.org/wiki/Heart.

[18] Wikipedia. *Cardia Muscle*. URL: https://en.wikipedia.org/wiki/Cardiac_muscle.

[19] Beau Pontre. *Medsci 737 Lecture 7*.

[20] Ariel H. Curiale, Flavio D. Colavecchia, Pablo Kaluza, Roberto A. Isoardi, and German Mato. "Automatic Myocardial Segmentation by Using A Deep Learning Network in Cardiac MRI". In: *arxiv* (2017). URL: https://arxiv.org/pdf/1708.07452v1.pdf.

[21] Phi Vu Tran. "A Fully Convolutional Neural Network for Cardiac Segmentation in Short-Axis MRI". In: *arxiv* (2017). URL: https://arxiv.org/pdf/1604.00494v3.pdf.

[22] Le Zhang et al. "Automated Quality Assessment of Cardiac MR Images Using Convolutional Neural Networks". In: *Springer* (2016). URL: https://link.springer.com/content/pdf/10.1007%2F978-3-319-46630-9_14.pdf.

[23] Zhaohan Xiong, Vadim V. Fedorov, Xiaohang Fu, Elizabeth Cheng, Rob Macleod, and Jichao Zhao. "Fully Automatic Left Atrium Segmentation from Late Gadolinium Enhanced Magnetic Resonance Imaging Using a Dual Fully Convolutional Neural Network". In: *ieee* (2018). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8447517.

[24] Shafiullah Soomro, Farhan Akram, Asad Munir, Chang Ha Lee, and Kwang Nam Choi. "Segmentation of Left and Right Ventricles in Cardiac MRI Using Active Contours". In: (2017). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5591936/pdf/CMMM2017-8350680.pdf.

[25] Qiao Zhenga, Herve Delingette, Nicolas Duchateau, and Nicholas Ayache. "3D Consistent and Robust Segmentation of Cardiac Images by Deep Learning with Spatial Propagation". In: *Arxiv* (2017). URL: https://arxiv.org/pdf/1804.09400.pdf.

[26] Nicolas Basty and Vicente Grau. "Super Resolution of Cardiac Cine MRI Sequences Using Deep Learning". In: *Springer* (2018). URL: https://link.springer.com/chapter/10.1007%2F978-3-030-00946-5_3#enumeration.

[27] Jiuxiang Gu et al. "Recent Advances in Convolutional Neural Networks". In: *arxiv* (2017). URL: https://arxiv.org/pdf/1512.07108v5.pdf.

[28] Olivier Bernard et al. "Deep Learning Techniques for Automatic MRI Cardiac Multi-structures Segmentation and Diagnosis: Is the Problem Solved?" In: *ieee* (2018). URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8360453.

[29] Qi Dou, Cheng Ouyang, Cheng Chen, Hao Chen, and Pheng-Ann Heng. "Unsupervised Cross-Modality Domain Adaptation of ConvNets for Biomedical Image Segmentations with Adversarial Loss". In: *arxiv* (2018). URL: https://arxiv.org/pdf/1804.10916.pdf.

[30] National Electrical Manufacturers Association. *Dicom*. URL: https://www.dicomstandard.org/.

[31] The HDF Group. *HDF5*. URL: https://www.hdfgroup.org/.